ICASE

RASTER-SCAN HIDDEN SURFACE ALGORITHM TECHNIQUES

Griffith Hamlin, Jr.

C. William Gear

Report Number 77-1

January 7, 1977

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING

NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

RASTER-SCAN HIDDEN SURFACE ALGORITHM TECHNIQUES

Griffith Hamlin, Jr.

C. William Gear*

ABSTRACT

Two new techniques are presented for reducing the number of depth calculations in hidden surface elimination. Two new algorithms using the techniques are compared with three existing algorithms and it is shown by examples that the new techniques reduce the number of multiplications involved in the depth calculations. A technique for increasing the parallelism of operations is also presented. This allows the calculation to be done more rapidly in hardware and is particularly useful for generating line drawings rather than the usual TV raster scan images in the common raster-scan hidden surface algorithms.

* University of Illinois at Urbana

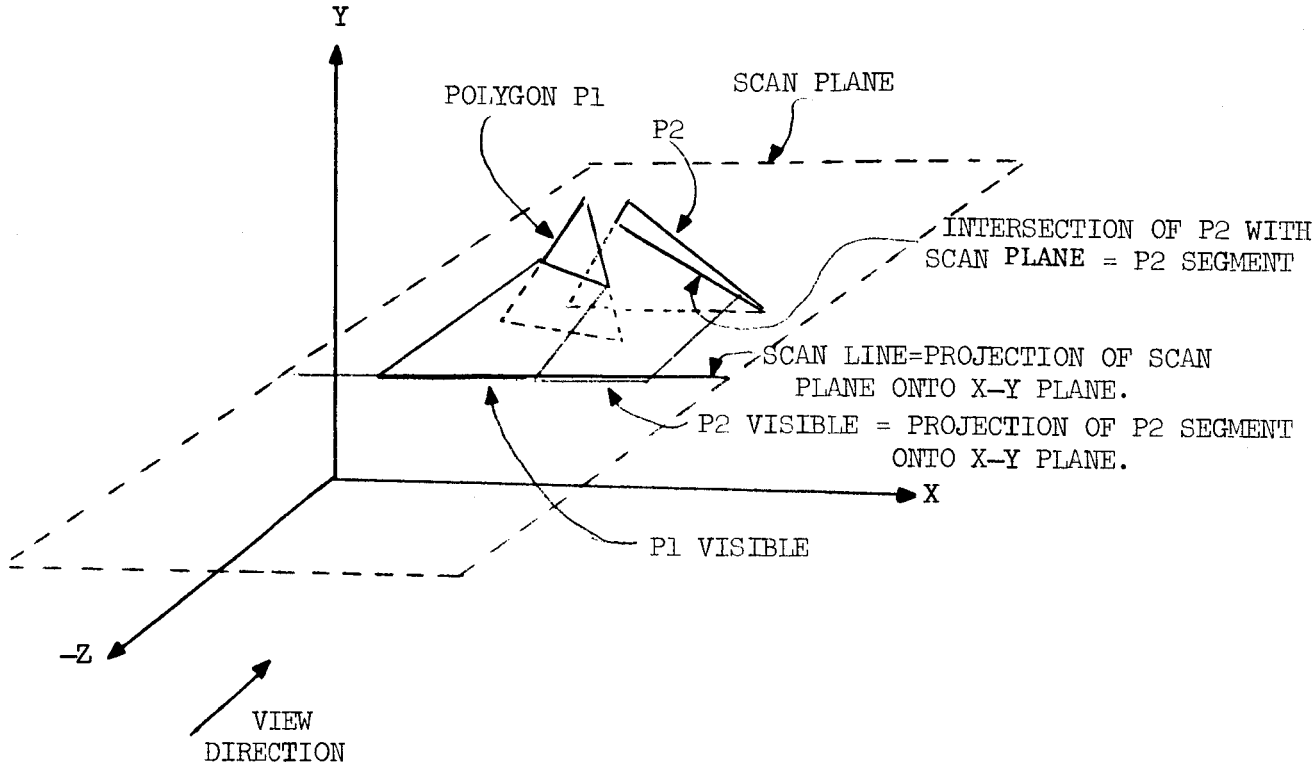
This report was prepared as a result of work performed under NASA Contract Number NAS1-14101 while both authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

I. INTRODUCTION.

Many algorithms for hidden surface elimination use a form of raster scan in which the scene is scanned by a sequence of horizontal planes in order to generate the projection of the information in each plane onto the corresponding scan line as shown in FIGURE 1. The advantage of this scheme is that it allows for fast sorting techniques to be used to preprocess the information into the order in which it will be needed in the display computations.

In all of the schemes to be discussed the scene is composed of a number of plane convex polygons in three dimensions. The two new techniques described assume that these polygons are non-intersecting, although both could be modified to compute intersections. All of the techniques discussed sort the vertices initially into ascending order on their Y coordinates. This means that as successive planes are processed in increasing Y order, the new polygon edges that intersect the current scan plane are the edges incident on the next vertices in the ordered vertex list. Within each scan plane the algorithms keep track of the X coordinates of each polygon edge that intersects the plane and keep this list of active edges in ascending order of X coordinate. It is then possible to scan across from left to right (increasing X value) and determine the depth of each polygon, and hence determine which is in front. Most algorithms then exploit the property of scan line coherence which is the property that the relative depths of polygons do not normally change from one scan line to the next, and that, furthermore, the ordering in the active edge list does not change much from one scan plane to the next. Indeed, if it is assumed that polygons cannot intersect, then the relative depths cannot change, and the

RASTER SCAN



TOP VIEW

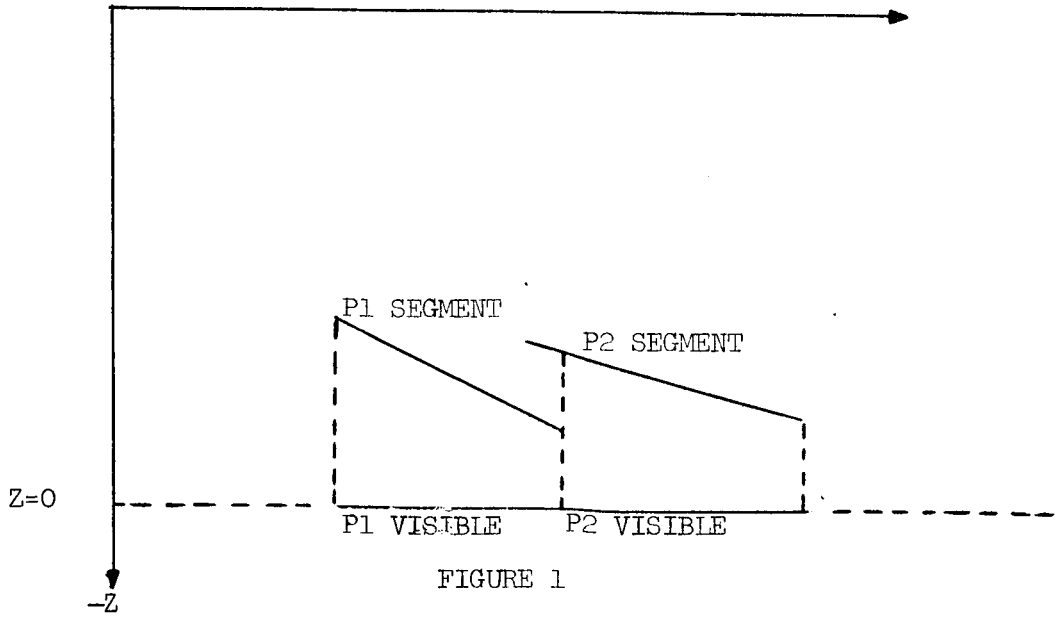


FIGURE 1

only times when it is necessary to perform new depth calculations are when the order in the list of active edges changes. This non-intersecting polygon assumption is reasonable in many environments. For example, the polygons might represent the surface of a solid body, such as an aircraft. Since two solid bodies cannot both occupy the same space, polygons on their surfaces could not intersect, although they might share sides in common.

The two new techniques introduced below use the following ideas:

1. In each scan line, a recursive method is used to process the polygons which insures that overlapping polygons are processed in order of the distance from the observer (decreasing Z). A stack is used to keep track of the polygons which were visible but have been obscured by closer polygons, and the polygons which are visible are numbered in such a way that the relative depth of any two polygons which are compared will be remembered for all future scan planes.

2. When the order of the active edge list changes, the visibility of polygons after the change can often be deduced from the visibility before the change and an examination of the type of the edges involved in the change. All edges are classified by marking them as left or right depending on whether they are on the left or right side of a polygon as seen by the viewer. For example, when two right edges cross and initially the left one is visible below the intersection, it follows that the right one is visible and the left invisible above the intersection. This technique is further enhanced by keeping track of those edges that are part of two or more polygons. Edges which are both left and right are classified as middle edges.

These two techniques have been programmed, and the two algorithms are described below. Both techniques are useful for any scene described by planar convex polygons, but the first is probably better for scenes in which there are many unconnected polygons. The second appears to be superior when the scene is composed of curved surfaces approximated by adjoining planar polygons. Descriptions of aircraft fuselages, wings, etc., used for structural and aerodynamic analysis are examples of such pictures.

The technique that increases the parallelism of computation and aids in the generation of line drawing is a technique that identifies the next largest Y for which any change occurs in the active edge list. Since this is the first scan plane at which a change in the visibility of polygons can occur, the visibility processing algorithm can advance immediately to that value and perform the new visibility computation. If a raster has to be developed, it can be generated in parallel with the new visibility computations. If a line drawing is to be generated, the lines can be output each time that they terminate or become invisible.

The next section summarizes the processing common to both algorithms. The following two sections describe the two algorithms in some detail. Section 5 discusses the technique for parallelism, and the final section presents some experimental results which compare the operation counts for several scenes ranging from 11 to 480 polygons.

2. BASIC ALGORITHM FEATURES.

The objective of any algorithm may be twofold. If a set of very different views of a scene are to be generated, then the objective is to reduce the total amount of computation. If, on the other hand, a series of views, each a small perturbation from the last as in a moving picture, are to be generated, then the objectives may include making as much of the process as parallel as possible in order to allow for several mini- or micro-processors to work on different sections. The objectives may also include organizing the calculation so that some of the work can be incremental, that is, so that small changes from one view to the next can be accomplished by a small amount of work.

The work is broken up into two stages: preprocessing and visibility computation. This not only allows for some parallelism, but also increases the possibility for incremental computation. Both algorithms do the rotation (and perspective transformation if desired) in the preprocessing stage. This can be performed incrementally. Next they sort the vertices by the Y coordinates. (The first technique only sorts the lowest vertex of each polygon.) For the first sort, a fast binary or bucket sort can be used, but subsequent incremental ones should use a variant of the bubble sort since changes in position will be few. Next, both algorithms compute coefficients A, B, and C for each polygon so that the polygon lies in the plane Z=AX+BY+C. If the polygon is 'edge on' to the viewer, it is dropped from further consideration. These coefficients can also be computed incrementally. Another step in preprocessing involves computing the slope of each polygon edge in the scene. This is computed as dx/dy which indicates the change in X in the movement from one scan plane to the next. (Horizontal lines require some special handling dependent on the details of the algorithm implementation.)

During the visibility computation, an Active Edge List is maintained in order of the X coordinates of the intersection of the edges with the current scan plane. When the scan plane is advanced by increasing Y to the next higher value, these X values must be updated by adding the inverse of the slopes of the edges to them. This may change the ordering so some testing must be done. The first algorithm described here saves information about scan line coherence by numbering the polygons. Consequently, the Active Edge List can be updated by simply resorting it. Since it is presumably almost sorted, a variant of the bubble sort provides a rapid method. The

second algorithm uses the information about which elements in the Active Edge List have changed positions, so it is more convenient to determine when two elements have to be exchanged and to perform the appropriate visibility computation immediately although it is conceptually similar to a bubble sort. In either case, the basic processing of one scan plane requires a number of additions equal to the number of active edges plus a like number of tests to determine if the order has changed. If there are any changes, additional computation is necessary. When the Y value of the next vertex is reached, a change must be made in the Active Edge List. This may involve adding, deleting or replacing edges. The details are algorithm dependent.

3. DESCRIPTION OF STACK ALGORITHM.

The first algorithm employs the first technique to reduce depth calculations. The input data describing the image consists of a set of planer, convex polygons, assumed to be opaque. These polygons may not pierce one another nor may they overlap one another cyclically. The polygons are numbered and sorted according to the first (lowest Y valued) scan line in which they appear. This produces the SORTED POLYGON LIST. At any scan line, an ACTIVE LIST is maintained of all polygons that are intersected by this scan line. Two new edges are added to the ACTIVE LIST from the SORTED POLYGON LIST on the first scan line in which they appear. Polygons are dropped from the ACTIVE LIST after processing the last scan line in which they appear. When the end of the SORTED POLYGON LIST is reached and the ACTIVE LIST is empty, the algorithm terminates.

Two entries are made in the ACTIVE LIST for each polygon, corresponding to the X-coordinates of the two points at which the scan ray enters and

exits the polygon as it scans the current scan line from left to right. Before each scan line is processed the ACTIVE LIST is sorted so the X-values are in left to right order.

Referring to FIGURE 2, the processing of an individual scan line begins with the first (leftmost) point in the ACTIVE LIST. The polygon entered at this point is declared to be the CURRENT polygon and procesing begins on it. To process a polygon means to scan the ACTIVE LIST data which lies within the left and right boundaries of the polygon, to determine when (if ever)

RECURSIVE PROCESSING

ONE SCAN LINE

TOP VIEW

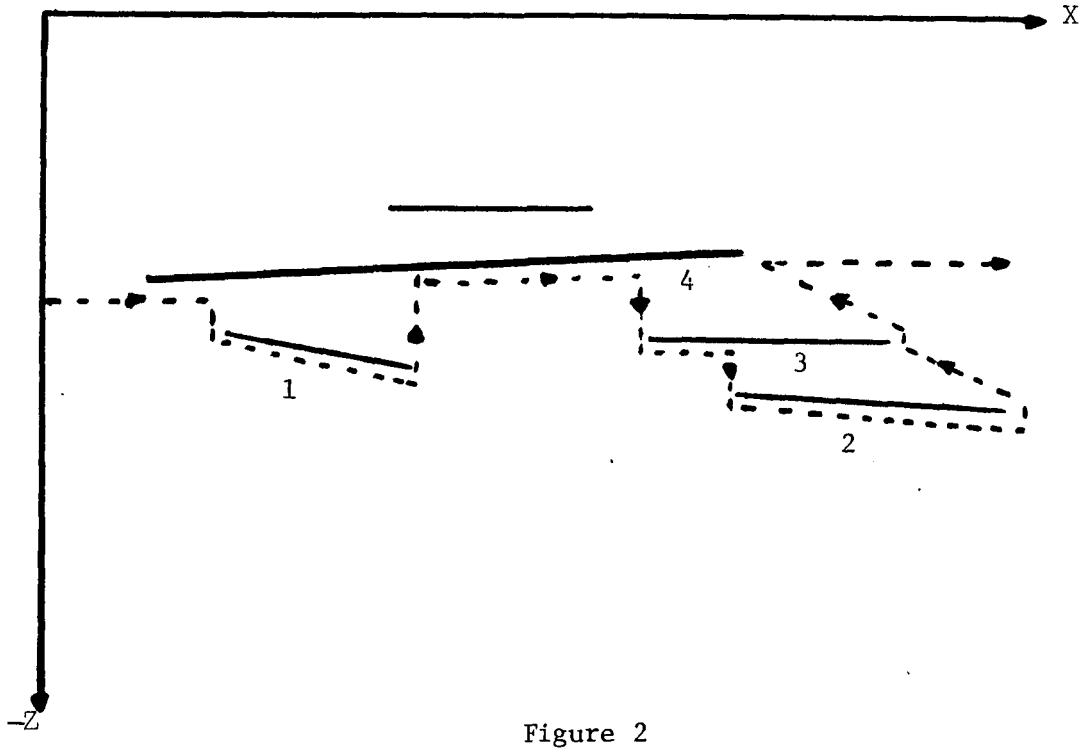


Figure 2

the polygon becomes visible, and to generate output data indicating the X-position on the current scan line where the polygon is visible. As the algorithm proceeds, a pointer into the ACTIVE LIST keeps track of the position of the scan ray as it advances from point to point.

When the scan ray encounters the exit of the CURRENT polygon, processing is terminated on that polygon. At this time the algorithm saves with the CURRENT polygon the ordinal number representing when the CURRENT polygon was processed (i.e., three is saved if the CURRENT polygon is the third polygon on this scan line for which processing has been terminated). This number will be called the rank of the polygon. Since the stacking mechanism causes the frontmost polygon of overlapping polygons to have its processing terminated first, the rank of two overlapping polygons may be used on the next scan line to indicate their relative depths. Ranks of two non-overlapping polygons are meaningless. Therefore, before using ranks, the algorithm checks data stored during processing of the previous scan line to insure that the two polygons overlapped on the previous scan line. After ranking a polygon, the stack is popped and processing is resumed on the polygon saved at the top of the stack. It now becomes the CURRENT polygon. When this occurs the scan ray and Active Edge List pointer jump back to the position they had when the polygon was pushed onto the stack. This may cause the algorithm to deviate from a strict left to right processing order. The algorithm remembers the rightmost X-value for which output has been produced. If, after a pop of the stack, the scan ray jumps to the left, no output is generated until the scan ray again reaches this rightmost value.

If, before exiting the CURRENT polygon, the scan ray encounters another

polygon which hides the CURRENT polygon, processing of the CURRENT Polygon is suspended, the Active Edge List pointer is pushed onto a stack, and processing is started on the newly encountered polygon, which has become visible.

If, before exiting the CURRENT POLYGON, the scan ray encounters the right side (exit) or a polygon, the algorithm marks that polygon. If that polygon is at all visible, it will have been pushed onto the stack. When it is later popped off the stack the mark will cause the algorithm to immediately rank it and pop another polygon from the stack.

If, on the other hand, the polygon is completely invisible, it will not have been pushed onto the stack. In this case the mark wil have the effect of deleting the polygon from further processing on the current scan line. This eliminates some Z-depth calculations involving this polygon. Instead of ranking such invisible polygons, the algorithm saves with each of them the number of the polygon that was visible when the invisible polygon was entered by the scan ray. This information may save Z-depth calculations on subsequent scan lines.

4. DESCRIPTION OF ALGORITHM CROSS

Algorithm CROSS employs the second technique. In this algorithm the fundamental elements in the representation of the data are the vertices. Each vertex consists of its coordinates X, Y, and Z plus some pointer information generated in the algorithm. Edges consist of a pair of vertices plus information generated in the algorithm. Polygons consist of an (ordered) list of edges. Polygons must be planar, convex, and

non-intersecting (although the algorithm could be modified to compute intersections). The usual preliminary processing is used to compute the coefficients of the planes of the polygons, to order the vertices by their Y-values, and to construct lists of edges connected to each vertex. In this case, edges are said to begin on a vertex if their end with the lowest Y value is on that vertex. They end on the vertex with the highest Y value. Two lists are constructed for each vertex, a list of those edges ending on a vertex and a list of those beginning on a vertex. The list of beginning edges is ordered by the slope of the edges so that at higher Y values the edges will be in a left to right order. The list of ending edges is unordered. It is used to remove edges from the Active Edge List when the vertex is encountered. This saves testing each active edge to see if it has ended.

4.1 EDGE CLASSIFICATION

Because edges can be 'shared' between two or more polygons, it is possible for an edge to be the left or right edge of more than one polygon. However, it is convenient if we associate a unique left and right polygon with each edge. Therefore, the algorithm creates enough copies of edges so that no edge is a left or right edge of more than one polygon. It is also necessary to avoid making more than one copy of an edge a 'middle' edge (because the non intersecting hypothesis could be violated). During part of the preprocessing, the edges are classified as L, M or R and copies are created if necessary. At the same time, a pair of pointers is created for each edge indicating its unique left and right polygons. At this time, the left polygon of a left edge is null, and similarly for the right polygon of a right edge. During the scanning, these unused positions will be taken

over for storing pointers to the next visible polygons when the edge is active and visible. Thus, for any active visible edge, the left pointer will always contain the polygon on the left which is visible, and similarly for the right pointer.

When an edge is put in the Active Edge List, pointers are stored with the polygon so that any polygon under the scan line has pointers to its left and right edges. (This is the reason for the restriction to convex polygons.)

The list of all edges under the current scan line is kept in left to right order. The X position of each edge is recorded, along with the edge type (L, M, or R), its visibility (V = visible, I = invisible) and pointers to the left and right polygons mentioned above. The list must be updated each time a vertex is encountered or two active edges cross.

4.2 VERTEX PROCESSING

When a vertex is encountered all edges ending on the vertex are removed. The vertex can be marked visible if any of these edges are visible, otherwise it is invisible. Its position in the Active Edge List is given by the position of the removed edges. If there are no edges to remove, the visibility of the vertex and its position in the active edge must be calculated. Its position is determined by searching through the active edge list sequentially. Its visibility is determined by computing the depth of the visible polygon at that point. (This polygon is known by looking to the left or right in the list.)

Next, all edges begining at the vertex are added to the Active Edge List and the crossing points to their left and right are calculated if necessary. If the vertex is invisible, there is nothing else to do. If it is visible, the visibility of each new edge must be computed. This is necessary because several polygons could start from a vertex, and these could obscure some of the edges. The visibility is determined by comparing the edge depth with the depth of the currently visible polygon on the left. If the visible polygon does not pass through the vertex under consideration, this requires a straightforward depth calculation and comparison. If the polygon does pass through the vertex, the comparison is done using the coefficients of the polygons and the slope of the added edge. (The latter is necessary to avoid the problem caused by the fact that both the polygon and the edge pass through the vertex under consideration.)

Anytime a middle edge is added its visibility is determined by seeing if its left polygon is the currently visible polygon. In addition, when a visible right edge is added, a search must be made to find the next visible polygon on the right. This is done using the search described in the next section, except when this is the last edge from the vertex. In that case, the next visible polygon can be determined by looking to the right in the Active Edge List.

4.3 CROSSING ANALYSIS

The principal feature of the visibility computation is the classification of crossing types. Each active edge is in one of the six states LV, MV, RV, LI, MI, or RI. This gives a total of 36 possibilities for the crossing of a pair of adjacent edges. These are shown in Table I

below. (The first edge is the edge with the smaller X value.) A pair of edges that cross are handled by switching their positions in the active edge list and then executing the action described in table below. For example, when an RV edge crosses an MV edge, the edges switch and the middle edge becomes invisible. The only other action needed is to update the right polygon pointer of the RV edge. Its new value is the right polygon pointer of the middle edge.

Table I

2nd Edge / 1st Edge	LV	MV	RV	LI	MI	RI
LV	I1	E	E	C1a	C1b	C1c
MV	I1	E	E	-	-	-
RV	C2	I2	I2	-	-	-
LI	-	-	C3c	-	-	-
MI	-	-	C3b	-	-	-
RI	-	-	C3a	-	-	-

Blank entries indicate no action, otherwise:
- I1 - Make first edge invisible
- I2 - Make second edge invisible
- C1 - If the second edge is in front of the polygon visible to the left of the first edge, make second edge visible. (Cases a, b, and c are distinguished later).
- C2 - Compares the depths of the polygons to the left and right of the first and second edges respectively. The one with the forward polygon remains visible.
- C3 - The same as C1 with first and second, and left and right interchanged.
- E - 'Error' condition because the edges belong to the same polygon. The crossing can be ignored.

The left polygon pointer of a left edge that is visible after the crossing calculation (and similarly for right edges) may have to be updated. In all but two cases, the pointer is available in the other edge.

In these two cases, C1c and C3c it is necessary to search to find the polygon that is now visible between the two edges. This search is done by processing the active edge from left to right looking for the edge under consideration while recording entries and exits from polygons. When the edge under consideration is reached, it is known which polygons span the intersection point. The depths of all of them must be calculated, and the most forward chosen as the new visible polygon.

Thus, of the 32 cases that can occur in Table I, 21 of them require no action, 4 of them require a simple change of visibility and the changing of a pointer, and 7 cases require a depth comparison of two polygons. In addition, two of the last seven cases require a search and possibly several depth calculations in those cases that the visibility changes.

4.4 PREPROCESSING

In addition to the preprocessing described in section 2, this algorithm has to do the following:

Edge Process

Each edge is examined. Its end points are placed in ascending order and a pair of chains is constructed from each vertex through the edges so that from each vertex it is possible to find the set of edges that end on the vertex and the set of edges that start on the vertex. The set of edges that start on the vertex are sorted in ascending order of their slopes so that they will be in the correct left to right order in the Active Edge List when they are entered there.

Polygon Edge Label

Each polygon is examined in turn. If it is close to 'edge-on' it is

ignored, otherwise the coefficients A, B, and C such that Z=AX+BY+C are calculated. This is done by solving a system of three linear equations derived from three adjacent vertices on the polygon. If the determinant of the system is too small, the first vertex is dropped and another added. This is repeated until a solution is found or the polygon is ignored as 'edge-on' or too small. If a solution is found, the determinent of the system (which is available as a byproduct) indicates whether the polygon is being processed in the clockwise or counter-clockwise direction. This information is used in two ways. The first is an option to ignore rear polygons. If this option is used, polygons must be specified in clockwise order when viewed from the exterior of the body. A counter-clockwise projection in the x-y plane indicates that the polygon should be dropped. If the polygon is not dropped, its edges are examined in sequence. The combination of the direction of rotation and the direction of the edge (up or down) indicates whether the edge is a left or right edge. It is marked accordingly in the list of edges, and the 'name' of the polygon on the left or right of the edge is stored in the edge list. If the edge is already marked as being of the same type, a duplicate edge is created. The edge of the rear polygon is saved as the duplicate edge (which is distinguishable in the edge list), so that the picture generation can simply ignore visibility tests on duplicate polygons. The front edge is easily found by finding the edge attached to the polygon with the largest value of the parameter A in the case of left edges, or the smallest value in the case of right edges. If the edge has been marked as of the other type (e.g. R when the new type is L) it is simply marked as M.

5. PARALLEL GENERATION OF RASTER AND LINE DRAWING TECHNIQUE

Since there is no visibility change from one scan line to the next until a new vertex is encountered or until two edges in the Active Edge List change places, the Active Edge List can be updated from one scan line to the next by additions only. There is no need to test to see if there have been any changes of position. Indeed, there is no need to generate the state of the Active Edge List for each scan plane, only for those on which a change occurs. The Y value of the next change is calculated by maintaining a list of all intersections of adjacent edges in the Active Edge List unless those edges were incident on the same vertex or were part of the same polygon. This list is maintained in order of the Y value of the intersection. Consequently the next Y value of a change can be found from the minimum in this list and the next entry on the sorted vertex list generated in preprocessing. It was found that the amount of arithmetic involved in computing the intersections was less than the amount of work involved in additions and tests used to generate successive scan plane Active Edge Lists. In this technique, a separate processor can be generating the state of the Active Edge List for each scan line by additions and generating the raster output from that state. Such a display generator was described by Erdahl[1] in 1972. It appears that algorithm CROSS produces the proper type of input required by Erdahl's display processor, e.g., a sequence of visible edge segments sorted by the initial scan line on which they appear and then sorted by the X-values of the intersection with the scan line. The visibility computation can proceed by advancing to the Y value of the next change. As implemented, even the X values of the active edge were not updated except when there was a change of visibility. This was partly in

order to allow line drawings to be generated easily, but also because it was found that the X coordinates of active edges were not needed very often, so that the additional work in calculating intersections from the data available was more than offset by the reduction in the number of times that the calculation of the X value had to be made. Each active edge contained its last calculated X value and the corresponding Y value in the implementation.

6. COMPARISON OF SEVERAL ALGORITHMS

The two algorithms described here, along with three other scan-line algorithms [2,3,4] were implemented in FORTRAN on a PRIME-300 minicomputer. Each of these implementations counted the number of tests, stored data accesses, multiplications or divisions, and additions required by the algorithm. Table 2 gives these counts for the several pictures shown in the appendix. The counts included all processing done on each individual scan line (the inner loop of all of these algorithms), the processing done in keeping the list of active polygons in left-to-right order, and the preprocessing done in calculating the coefficients of the plane of each polygon. The counts do not include other preprocessing done in reading the input polygons or sorting them in order of their topmost vertex. This Y-sort is performed by all algorithms tested and so could be omitted from the counts. In assigning counts to each portion of each algorithm it was assumed that a moderate amount of data could be placed temporarily in registers with essentially instantaneous access. For this reason no storage accesses are included for incrementing indices and other control pointers. In assigning additions and multiplications, no additions to program loop counters or other such programming functions were included. Such operations

are very much implementation dependent, not an integral part of the algorithm, and so were omitted entirely. It was difficult to collect data on our FORTRAN version of the Watkins algorithm that could be compared directly with the other algorithms. The Watkins algorithm was designed to handle intersecting polygons which the other algorithms do not handle. Our implementation simplified the Watkins algorithm by removing this feature. Also, the Watkins algorithm was designed to accept input vertices with a precision greater than one raster point. Our input data contained vertices that were rounded off to the nearest raster point. This sometimes caused the Watkins algorithm to think that two adjacent polygons overlapped (by one raster unit), causing some extra calculations.

For all pictures combined, the two algorithms described here performed fewer multiplications and additions than any of the three algorithms obtained from the literature. In the case of algorithm STACK, this was accomplished at the expense of increased testing of previously stored data.

Algorithm CROSS can be used for raster or line drawings, and makes efficient use of connectivity information. Consequently it appears to be faster than many other techniques. However, it uses considerably more storage than other techniques because of the large number of pointers. (The present implementation uses about 40 16-bit integers and 12 32-bit floating point values per 4-sided polygon. However, it is an experimental version with an unnecessary number of pointers to allow any variation to be tried. A better implementation could reduce this to approximately 22 16-bit integers and 9 32-bit floating point values per 4-sided polygon.)

A major problem with algorithm CROSS is due to the fact that it does use the connectivity information effectively. Once an error in visibility is made, it can propagate through the figure. This causes particular problems if two vertices are placed in the same physical location, or close enough that round-off error will take its toll. If the algorithm is told that two points are the same, the connectivity and slopes of the polygons can be used to determine visibility, otherwise a simple depth calculation must be made, and round-off errors could yield the wrong conclusion. Discarding very small polygons helps with this problem. The alternative is to coalesce very close points and give them the same 'name' in the data structure.

References

[1] Erdahl, Alan C., "Displaying Computer Generated Half-tone Pictures in Real Time," Computer Science Department, University of Utah, RADC-TR-69-250, 1972.

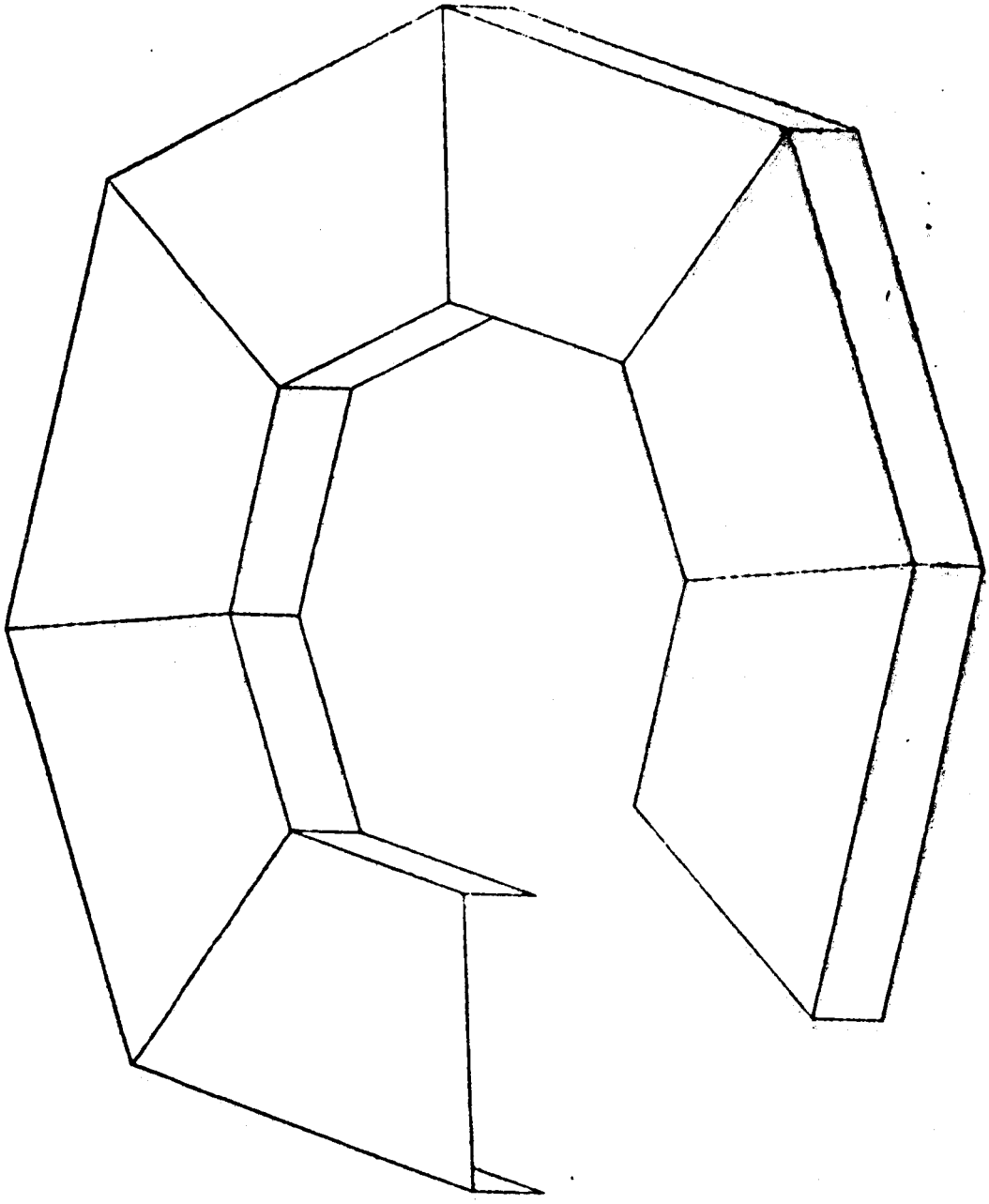
[2] Bouknight, W. J., "A Procedure for Generation of Three-dimensional Half-toned Computer Graphics Presentations," Communications of the ACM, Vol, 13, No. 9, September 1970, pp. 527-536.

[3] Romney, G. W., G. S. Watkins, D. C. Evans, "Real-time Display of Computer Generated Half-tone Perspective Pictures,' IFIP, 1968, pp. 973-978.

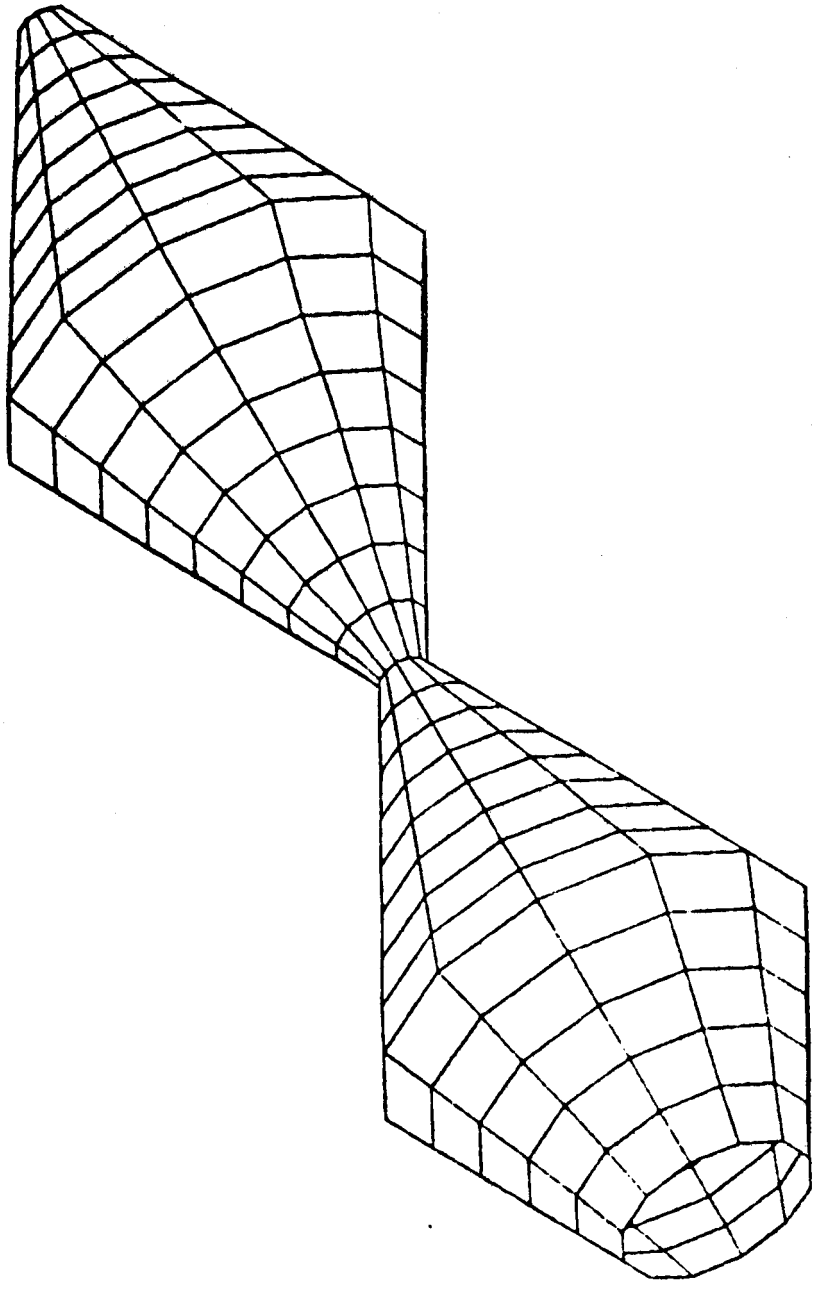
[4] Watkins, G. S., "A Real-time Visible Surface Algorithm," Computer Science Department, University of Utah, UTECH-Csc-70-101, June 1970.

Algorithm	Memory	Tests	Mult/Div	Additions
			CHANNEL PICTURE	
STACK	53983	27701	550	5352
CROSS	14471	3882	377	2922
Bouknight	50175	16534	10008	14142
Romney	52290	13388	1114	5248
Watkins	85641	64385	15537	35817
			CONE PICTURE	
STACK	259686	131112	9783	23752
CROSS	79723	22680	5639	8753
Bouknight	242524	85682	48787	61471
Romney	261167	91720	36233	48917
Watkins	345671	262095	54599	130615
			CYLINDER PICTURE	
STACK	298237	156147	7767	23913
CROSS	86229	22090	4000	11950
Bouknight	273648	95549	55295	71173
Romney	291893	99956	36569	52447
Watkins	382588	308619	72607	169599
			BOTTLE PICTURE	
STACK	70503	36788	1289	5459
CROSS	26415	7032	713	4820
Bouknight	71787	23949	14501	19249
Romney	75012	22320	4979	10055
Watkins	121989	97624	24431	56155
			AIRCRAFT PICTURE	
STACK	266557	131252	13180	25672
CROSS	115515	31429	7506	12489
Bouknight	244130	89343	44976	55756
Romney	260708	93903	32666	43446
Watkins	358953	252930	56236	130254
			SQUARES PICTURE	
STACK	41041	19712	263	3816
CROSS	18996	5014	223	3207
Bouknight	39599	12921	5963	9431
Romney	43984	11753	723	4191
Watkins	108840	89533	30710	67078

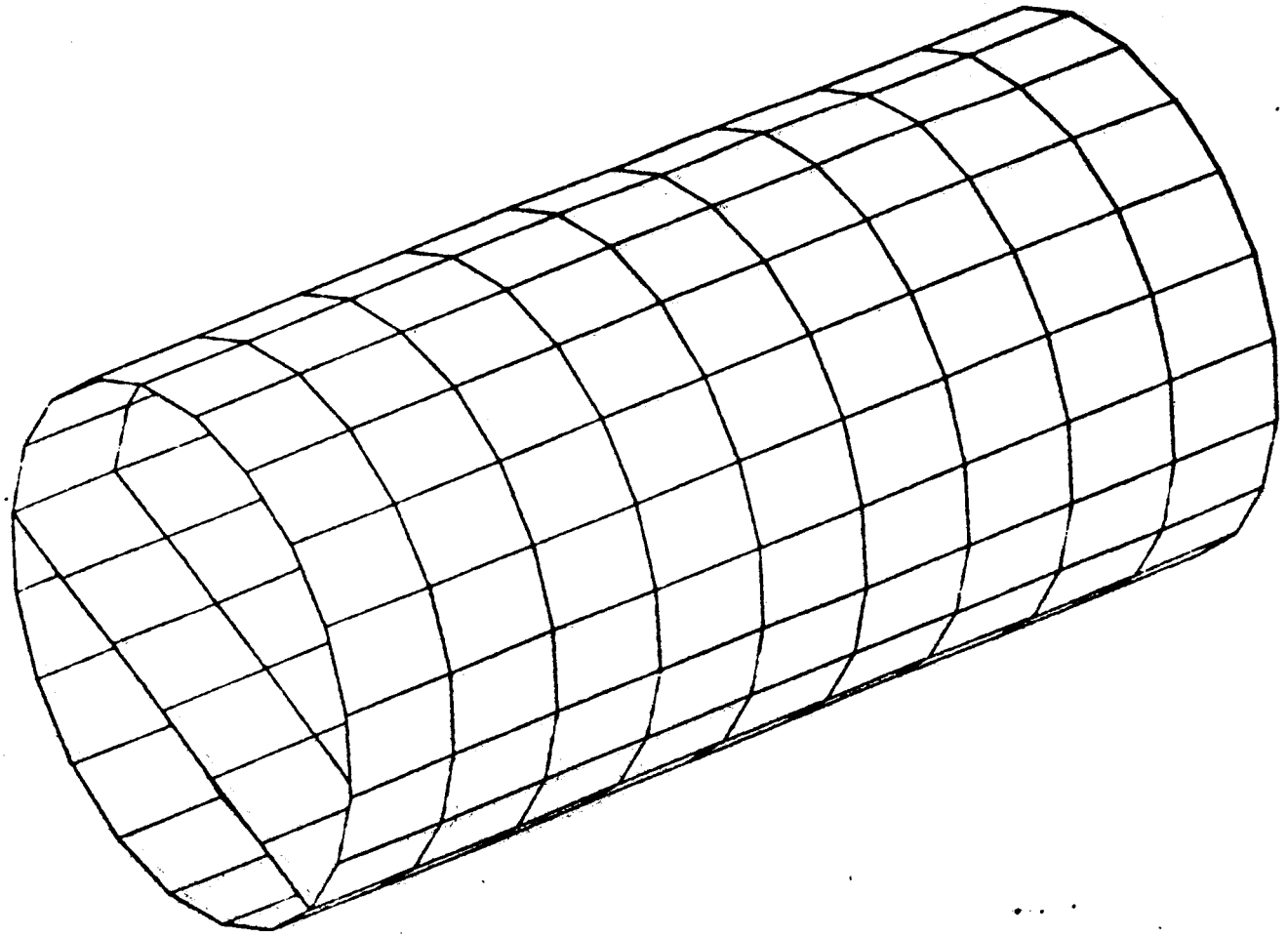
TABLE 2.



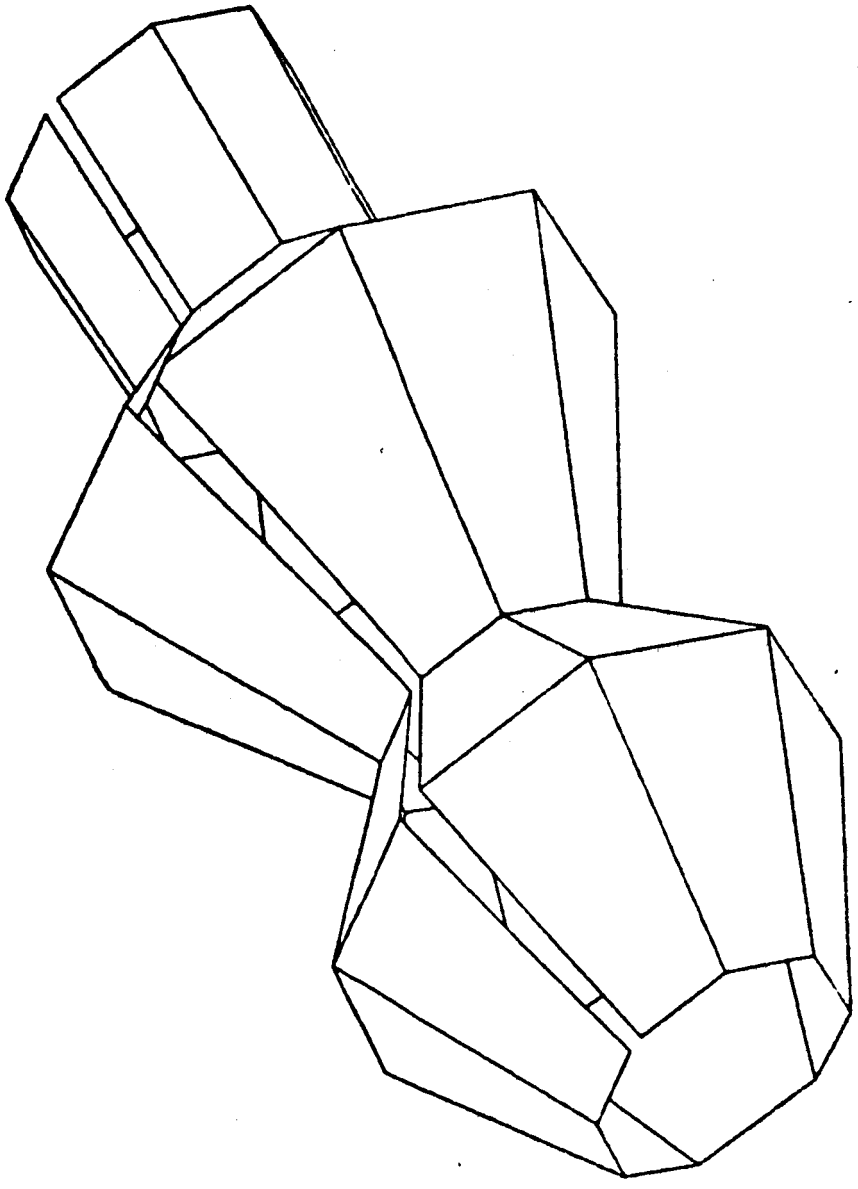
CHANNEL



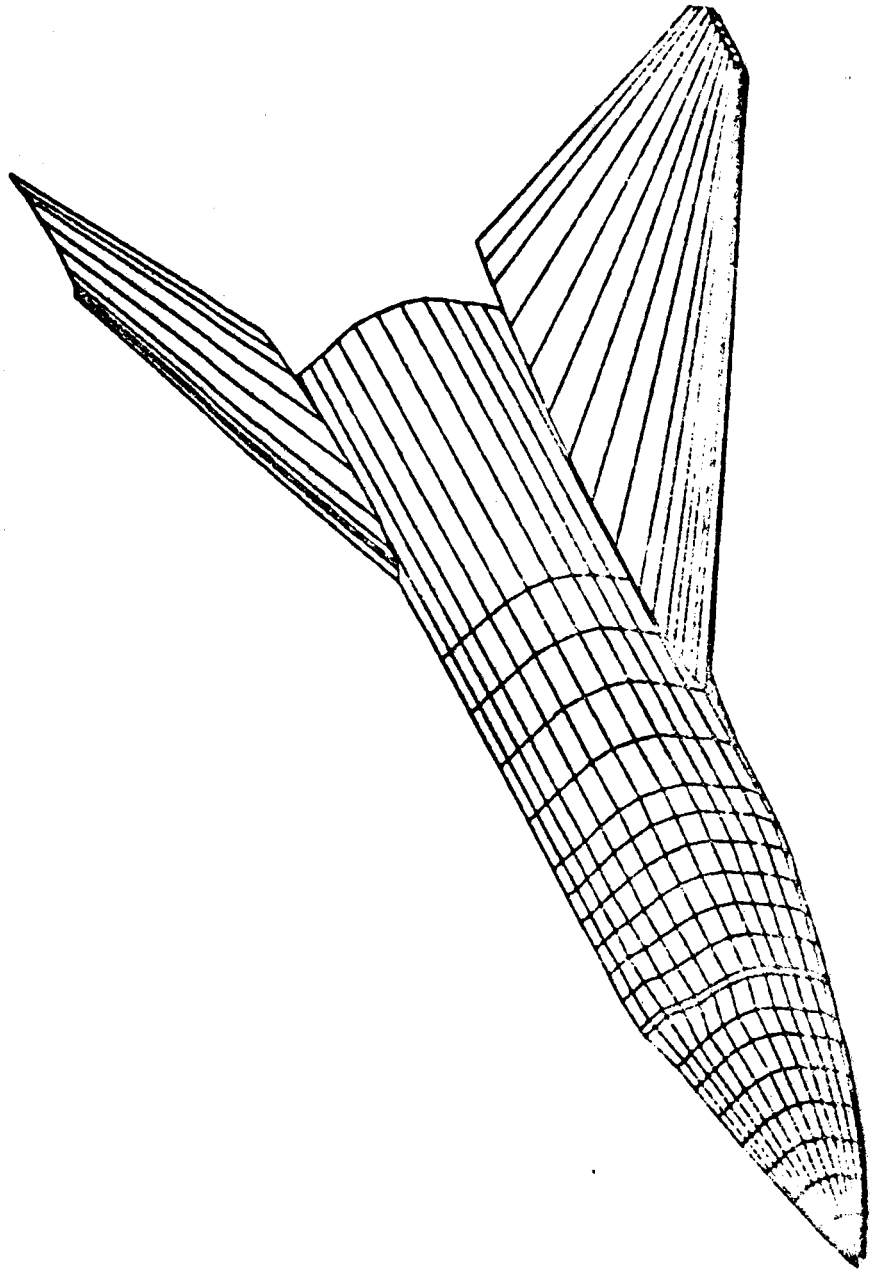
CONE



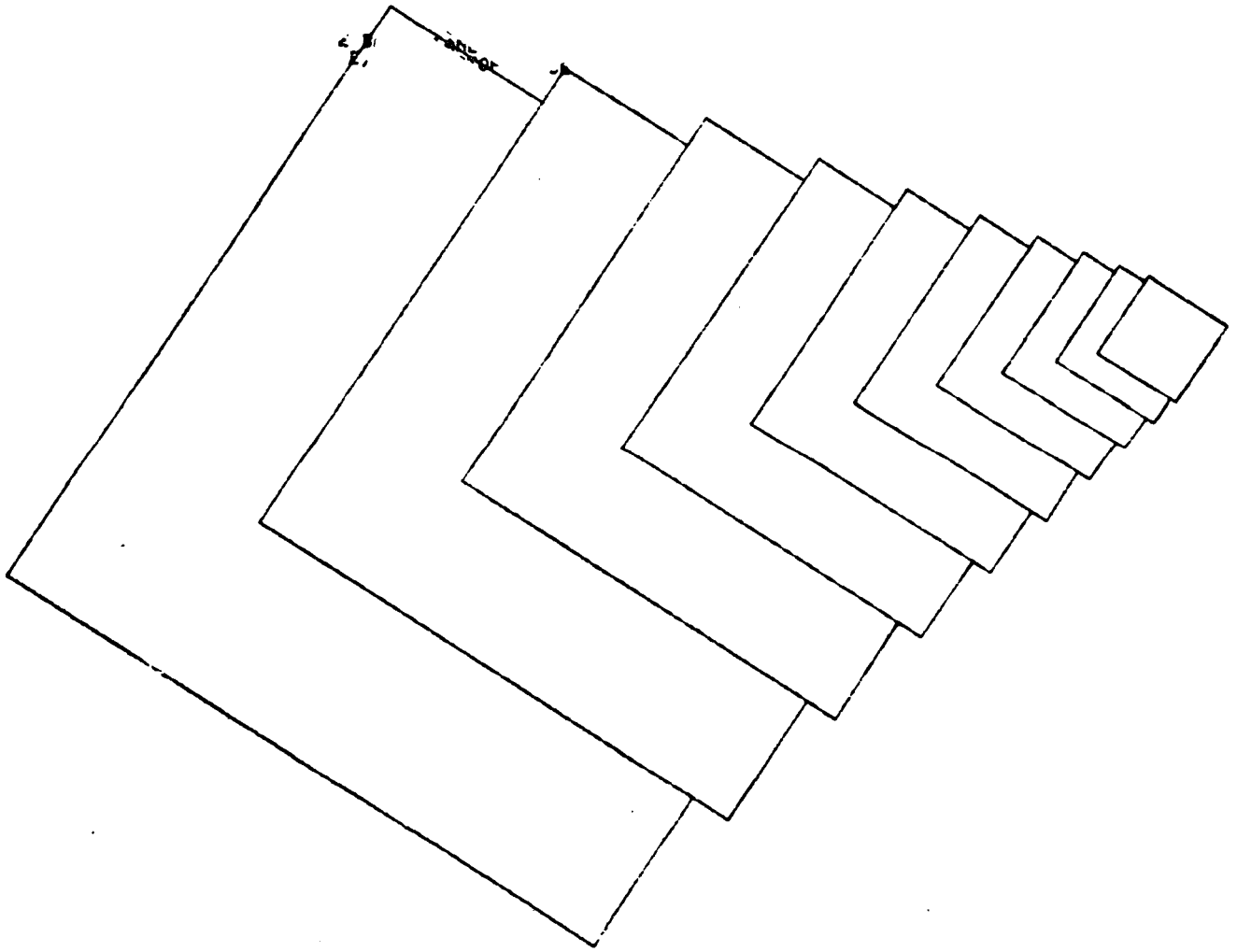
CYLINDER



BOTTLE PICTURE



AIRCRAFT



SQUARES